001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017package org.apache.commons.collections4.iterators;  
018  
019import java.util.NoSuchElementException;  
020  
021import org.apache.commons.collections4.ResettableIterator;  
022  
023/\*\*  
024 \* <code>SingletonIterator</code> is an {@link java.util.Iterator} over a single  
025 \* object instance.  
026 \*  
027 \* @since 2.0  
028 \*/  
029public class SingletonIterator<E>  
030 implements ResettableIterator<E> {  
031  
032 /\*\* Whether remove is allowed \*/  
033 private final boolean removeAllowed;  
034 /\*\* Is the cursor before the first element \*/  
035 private boolean beforeFirst = true;  
036 /\*\* Has the element been removed \*/  
037 private boolean removed = false;  
038 /\*\* The object \*/  
039 private E object;  
040  
041 /\*\*  
042 \* Constructs a new <code>SingletonIterator</code> where <code>remove</code>  
043 \* is a permitted operation.  
044 \*  
045 \* @param object the single object to return from the iterator  
046 \*/  
047 public SingletonIterator(final E object) {  
048 this(object, true);  
049 }  
050  
051 /\*\*  
052 \* Constructs a new <code>SingletonIterator</code> optionally choosing if  
053 \* <code>remove</code> is a permitted operation.  
054 \*  
055 \* @param object the single object to return from the iterator  
056 \* @param removeAllowed true if remove is allowed  
057 \* @since 3.1  
058 \*/  
059 public SingletonIterator(final E object, final boolean removeAllowed) {  
060 super();  
061 this.object = object;  
062 this.removeAllowed = removeAllowed;  
063 }  
064  
065 //-----------------------------------------------------------------------  
066 /\*\*  
067 \* Is another object available from the iterator?  
068 \* <p>  
069 \* This returns true if the single object hasn't been returned yet.  
070 \*  
071 \* @return true if the single object hasn't been returned yet  
072 \*/  
073 @Override  
074 public boolean hasNext() {  
075 return beforeFirst && !removed;  
076 }  
077  
078 /\*\*  
079 \* Get the next object from the iterator.  
080 \* <p>  
081 \* This returns the single object if it hasn't been returned yet.  
082 \*  
083 \* @return the single object  
084 \* @throws NoSuchElementException if the single object has already  
085 \* been returned  
086 \*/  
087 @Override  
088 public E next() {  
089 if (!beforeFirst || removed) {  
090 throw new NoSuchElementException();  
091 }  
092 beforeFirst = false;  
093 return object;  
094 }  
095  
096 /\*\*  
097 \* Remove the object from this iterator.  
098 \*  
099 \* @throws IllegalStateException if the {@code next} method has not  
100 \* yet been called, or the {@code remove} method has already  
101 \* been called after the last call to the {@code next}  
102 \* method.  
103 \* @throws UnsupportedOperationException if remove is not supported  
104 \*/  
105 @Override  
106 public void remove() {  
107 if (removeAllowed) {  
108 if (removed || beforeFirst) {  
109 throw new IllegalStateException();  
110 }  
111 object = null;  
112 removed = true;  
113 } else {  
114 throw new UnsupportedOperationException();  
115 }  
116 }  
117  
118 /\*\*  
119 \* Reset the iterator to the start.  
120 \*/  
121 @Override  
122 public void reset() {  
123 beforeFirst = true;  
124 }  
125  
126}